Securing AES against Second-Order DPA by Simple Fixed-Value Masking

Hwasun Chang Kwangjo Kim

International Research center for Information Security (IRIS)
Information and Communications Univ. (ICU)
58-4 Hwaam-dong, Yuseong-gu, Daejon, 305-732, Korea.

{s1364, kkj}@icu.ac.kr

Abstract Since Differential Power Analysis (DPA) has been announced, many countermeasures and improved DPAs have been proposed for many algorithms. For securing AES, masking methods were proposed as countermeasures. But all the previous masking methods have been shown to be vulnerable to second order DPA (SODPA). We propose a masking method that is resistant to SODPA and more efficient than previous methods in respect to required memory and additional processing.

1 Introduction

Smart cards are used as tamper-resistant devices. But Kocher *et al.* [1] introduced DPA and it was shown that the key used internally by smart cards can be found by side channel attacks. After that, many countermeasures have been proposed and used. Messerges [2] proposed random masking method for securing AES finalists against power analysis. But the method uses random masks generated for each execution, and new S-boxes have to be computed accordingly. This causes much processing and much RAM is required to store modified S-boxes. Because RAM is a scarce resource in smart cards and processing power of smart cards is relatively low, this can be a problem especially for low cost smart cards. In the meantime, Clavier [3] showed that software countermeasures should be implemented even though hardware countermeasures exist. Nowadays many hardware countermeasures and software countermeasures are used simultaneously to protect secret data in smart cards. Recently, hardware countermeasure against side channel attacks especially against probing attacks was suggested [10].

To solve the problem of random masking method, two kinds of improvements were proposed. One is fixed value masking method and the other is multiplicative masking method. Fixed value masking method was suggested by Itoh *et al.* [4]. In the method, fixed masks and modified S-boxes are stored in ROM and a set of masks and a modified S-box is selected randomly in the beginning of encryption algorithm. Because modified S-boxes are computed in advance and stored in ROM instead of RAM, it requires less processing and uses less RAM than random masking method. And because ROM has relatively more space than RAM in smart cards, this method is practical. But this method is vulnerable to SODPA [8]. Multiplicative masking method was proposed by Akkar [5]. But Golic [6] showed that this method is inherently vulnerable to DPA. Some modifications of multiplicative masking method [6, 7] were proposed but they required much memory and processing.

It seems that fixed value masking method is the most suitable masking method especially for low cost smart cards. But it is vulnerable to SODPA. In this paper, we propose a method that uses less memory and processing than previous methods and which is resistant to SODPA.

2 Fixed-Value Masking

In this method, q sets of masks and the corresponding modified S-boxes are computed in advance and stored in ROM. Two types of fixed-value masking methods were proposed by Itoh *et al.* In the first method, the applied masks are same across rounds but they are different from byte to byte in intermediate results called the State. In the second method, the masks are same across rounds and bytes in the State.

When the encryption is executed, a set of masks and a modified S-box is selected randomly and used. Because second method uses less ROM, it is more adequate for low cost smart cards. AES algorithm using the second method is shown in Fig. 1 and **Algorithm 1**. **Algorithm 2** is the modified version of *ByteSub* transformation so that modified S-box stored in ROM can be used. *ShiftRow* and *MixColumn* are same with the original transformations.

There are three kinds of masks $FK_{i,r}$ used for masking round keys. That is for $i = 0$, $i = 1, \ldots, nr-1$, and $i = nr$ where nr is the number of rounds in AES. And these masks are derived from FIN and FOUT. FIN is the mask that the bytes in the State have before entering into *ByteSub_FM* transformation and FOUT is the mask that the bytes in the State have after *ByteSub_FM* transformation. FIN and FOUT are generated randomly so that any bit of $FK_{i,r}$ is 0 with probability 1/2. (1) shows how $FK_{i,r}$ is derived from FIN and FOUT. Modified S-boxes are computed by **Algorithm 3**.

Algorithm 1 $FixedMasking_{AESEnc}$

```
$FixedMasking_{AESEnc}(P)$
1       $/*\ K'_{i,r}$ : masked round key
        $(i = 0, 1, \ldots, nr, r = 0, 1, \ldots, q-1)\ */$
2       $r = GenerateRandomNumber();$
        $/*$ choose $r = 0, 1, \ldots, q-1/$
3       $T' = P;$
4       $for(i = 0; i < nr-1; i++)\{$
5           $T' = T' \oplus K'_{i,r};$
6           $T' = ByteSub\_FM(T', r);$
7           $T' = ShiftRow(T');$
8           $T' = MixColumn(T');$
9       $\}$
10      $T' = T' \oplus K'_{nr-1,r};$
11      $T' = ByteSub\_FM(T', r);$
12      $T' = ShiftRow(T');$
13      $T = T' \oplus K'_{nr,r};$
14      output $T;$
```

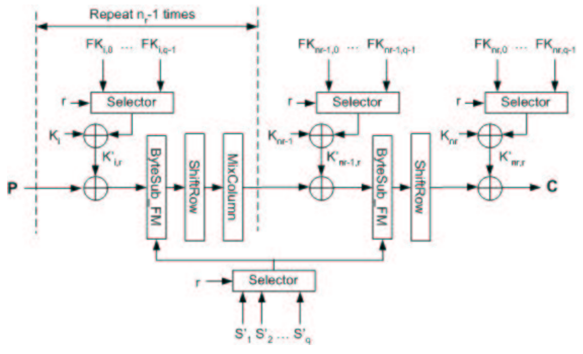

Figure 1: Fixed-value masking

Algorithm 2 *ByteSub_FM*

```
$ByteSub\_FM(X, r)$
1   $(x_{15}, x_{14}, \ldots, x_0) = X;$
2   $for(j = 0; j < 16; j++)\ x_j = S'_r[x_j];$
3   $X = (x_{15}, x_{14}, \ldots, x_0);$
4   output $X;$
```

Algorithm 3 *SboxUpdate*

```
$SboxUpdate(S, r)$
1   $for(x = 0; x < 256; x++)$
        $S'[x] = S[x \oplus FIN_r] \oplus FOUT_r;$
2   output $S';$
```

In **Algorithm 1**, T' is the intermediate masked value. Let K_i be the original round key and $FK_{i,r}$ be the fixed mask value. Then, $K'_{i,r}$ satisfies $K'_{i,r} = K_i \oplus FK_{i,r}$.

$$FK_{i,r} = \begin{cases} FIN_r \ (i = 0) \\ ShiftRow(MixColumn \\ (FOUT_r)) \oplus FIN_r \\ \qquad (i = 1, \ldots, nr-1) \\ ShiftRow(FOUT_r) \\ \qquad (i = nr) \end{cases} \tag{1}$$

3 Simple Fixed-Value Masking

We propose simple fixed-value masking method that requires less ROM and additional processing than previous fixed-value masking method. In the method, q pairs of one byte masks and modified S-boxes are stored in ROM. Modified S-box is computed using the corresponding mask as in **Algorithm 6**. One pair is selected randomly at the start of encryption. All bytes of plaintext are masked by the same one byte mask. Masked plaintext is encrypted using modified S-box. At the end of the algorithm, mask is applied once again and the result is the desired ciphertext. In the following figure and algorithms, m_r $(r = 0, 1, \ldots, q-1)$ represents the selected mask. r is the index for the chosen mask and modified S-box. S is the original S-box and S' is the modified S-box. *ByteSub_FM* is same with **Algorithm 2**.

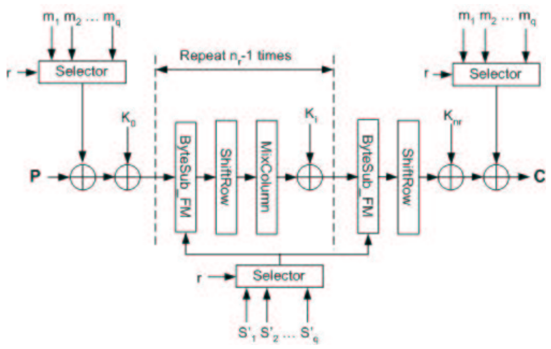

Figure 2: Simple fixed-value masking

Algorithm 4 $SimpleFixedMasking_{AESEnc}$

```
$SimpleFixedMasking_{AESEnc}(P)$
1    $r = GenerateRandomNumber();$
     $/*\ choose\ r = 0, 1, \ldots, q-1/$
2    $T' = P;$
3    $T' = ApplyMask(T', r);$
4    $T' = T' \oplus K_0;$
5    $for(i = 0; i < nr-1; i++)\{$
6        $T' = ByteSub\_FM(T', r);$
7        $T' = ShiftRow(T');$
8        $T' = MixColumn(T');$
9        $T' = T' \oplus K_i;$
10   $\}$
11   $T' = ByteSub\_FM(T', r);$
12   $T' = ShiftRow(T');$
13   $T' = T' \oplus K_{nr};$
14   $T = ApplyMask(T', r);$
15   $output\ T;$
```

Algorithm 5 *ApplyMask*

```
$ApplyMask(T', r)$
1    $(t'_{15}, t'_{14}, \ldots, t'_0) = T';$
2    $for(j = 0; j < 16; j++)\ t'_j = t'_j \oplus m_r;$
3    $T' = (t'_{15}, t'_{14}, \ldots, t'_0);$
4    $output\ T';$
```

Algorithm 6 *SboxUpdate2*

```
$SboxUpdate2(S, r)$
1    $for(x = 0; x < 256; x++)$
         $S'[x] = S[x \oplus m_r] \oplus m_r;$
2    $output\ S';$
```

3.1 Mask across the Transformations

The reason why the same mask is maintained across the processing is to use the modified S-box and the same mask for all bytes of the State. We will show that the mask is maintained for each transformation used in AES.

The mask is maintained by *RoundKeyAddition* because $(T'_{i,j} \oplus m_r) \oplus K_{i,j} = (T'_{i,j} \oplus K_{i,j}) \oplus m_r$ $(r = 0, 1, \ldots, q-1, i = 0, 1, \ldots, nr, j = 0, 1, \ldots, 15)$. $T'_{i,j}$ is the jth byte of the State and m_r is the selected mask. $K_{i,j}$ is the jth byte of the round key i. During *ByteSub* transformation the mask is maintained because the modified S-box is generated so that the mask does not change using **Algorithm 6**. *ShiftRow* transformation does not change the mask because all the bytes in the State have same masks and it changes only the position.
In *MixColumn* transformation, the columns of the State are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by $c(x) =' 03'x^3 +' 01'x^2 +' 01'x +' 02'$. This can be written as a matrix multiplication. Let $b(x) = c(x) \oplus a(x)$,

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (2)$$

If $a_i (i = 0, 1, 2, 3)$ is masked with m_r,

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \oplus m_r \\ a_1 \oplus m_r \\ a_2 \oplus m_r \\ a_3 \oplus m_r \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \oplus m_r \begin{bmatrix} 02 \oplus 03 \oplus 01 \oplus 01 \\ 01 \oplus 02 \oplus 03 \oplus 01 \\ 01 \oplus 01 \oplus 02 \oplus 03 \\ 03 \oplus 01 \oplus 01 \oplus 02 \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} b_0 \oplus m_r \\ b_1 \oplus m_r \\ b_2 \oplus m_r \\ b_3 \oplus m_r \end{bmatrix} \tag{5}$$

From the above equations, we can see that the mask before $MixColumn$ transformation is maintained after $MixColumn$.

3.2 Security Evaluation of Simple Fixed-Value Masking Method

To apply DPA, there should be an intermediate value that is a function of plaintext(or ciphertext) and part of encryption key. If this condition is met, we can find the encryption key part by part. But in the proposed method, all the bytes in the State are masked with randomly selected mask and an attacker cannot know which value is used for masking. Therefore, the attacker cannot know the bytes of the State from plaintext and part of key. Because the mask is maintained until the final round key addition finishes, the attacker cannot guess the bytes of the State before final round key addition from the ciphertext. If we choose the masks so that any bit of mask can be 0 with probability with 1/2, simple fixed-value masking method will be secure against probabilistic DPA presented in [4]. The proof will be similar to that of [4].

But this method is vulnerable to SODPA like other previous masking methods. We will update the method in the later part of this paper to make it secure against SODPA.

3.3 Advantage of Simple Fixed-Value Masking Method

The proposed method has advantage in memory and processing requirement. Previous fixed-value masking method requires at least three bytes to store a mask. That is, one for initial round key, another for intermediate round keys, and the third for final round key. But simple fixed-value mask method uses only one byte mask.

The simple fixed-value masking method requires less processing because mask is applied only at two points. That is, at the beginning and at the end of encryption. But the fixed-value masking method applies mask at $nr + 1$ points. Once at the initial round key addition, and $nr - 1$ times at intermediate round key addition and once at final round key addition. Moreover if the key becomes larger than 128 bits, it requires more exclusive-OR computation at one point than the simple version. For AES-128, the number of exclusive-OR operations for applying mask is like this. In fixed-value masking method, it is 11 points x 16 bytes = 176 operations. In simple fixed-value masking method, it is 2 points x 16 bytes = 32 operations. We summarize the comparison in Table 1.

Table 1: Comparison of Fixed-Value Masking (FVM) and Simple Fixed-Value Masking (SFVM) for AES-128

	FVM	SFVM
ROM space for mask	3 Bytes/mask	1 Byte/mask
Number of exclusive-OR operations for masking	176	32

3.4 Considerations for Security

If the masks and modified S-boxes are stored in ROM, the values are known to the developer and the values are same across smart cards that have been produced with same mask. Maybe some attacks could be tried using this fact. If the masks are stored in EEPROM and the masks and modified S-boxes are generated secretly and different from card to card, cards will be more secure.

4 Modification of Simple Fixed-Value Masking against Second Order DPA

Masking method is known to be subject to SODPA [8]. We propose a method for securing masking method against SODPA.

4.1 Second Order DPA (SODPA)

While masking method can be a countermeasure for first-order DPA, it is vulnerable to SODPA. Let's consider implementation of masking method as shown in **Algorithm 7**.

Algorithm 7 $MaskedFunction$

```
$MaskedFunction(T)$
1   $m = GenerateRandomNumber();$
2   $maskedT = T \oplus m;$
3   $T' = maskedT \oplus SecretKey;$
4   $\ldots$
5   $other\ operations$
6   $\ldots$
7   $output\ T';$
```

It is known that if there is a linear relationship between the instantaneous power consumption and the Hamming weight of the data being processed, **Algorithm 8** is sound. In **Algorithm 8**, i is the position of one bit in SecretKey in **Algorithm 7**.

Algorithm 8 $SecondOrderDPA$

```
$SecondOrderDPA(i)$
1   $for(b = 0; b <= 1; b++)\{$
2      $Calculate\ average\ statistics\ \overline{S_b} = |P_1 - P_3|\ by\ repeating\ the\ following\{$
3         $Set\ the\ i-th\ bit\ of\ T\ input\ to\ b.$
4         $Make\ remaining\ bits\ of\ T\ random.$
5         $Collect\ algorithm's\ instantaneous\ power\ consumption\ at\ lines\ 1\ and\ 3.$
          $Call\ these\ values\ P_1\ and\ P_3\ respectively.$
6      $\}$
7   $\}$
8   $Calculate\ the\ DPA\ bias\ statistic\ B = \overline{S_0} - \overline{S_1}$
9   $If\ B > 0\ then\ the\ ith\ key\ bit\ is\ a\ one,\ otherwise,\ it\ is\ a\ zero$
```

4.2 Modification of Simple Fixed-Value Masking Method

We can avoid SODPA by making it difficult for an attacker to decide the point where mask is loaded. For this, we can load more than two masks into memory that can be used for operand of exclusive-OR. And then we can use one mask among them randomly. SODPA resistant simple fixed-value masking method is shown in algorithm 9.

Algorithm 9 $SODPAResistantSFVM_{AESEnc}$

```
$SODPAResistantSFVM_{AESEnc}(P)$
1   $r[0] = GenerateRandomNumber();$
    $/*\ choose\ r = 0, 1, \ldots, q-1/$
2   $r[1] = GenerateRandomNumber();$
3   $m[0] = m_{r[0]};$
4   $m[1] = m_{r[1]};$
5   $l = GenerateRandomNumber();$
    $/*\ choose\ l = 0\ or\ 1\ */$
6   $T' = P;$
7   $T' = ApplyMask2(T', l);$
8   $T' = T' \oplus K_0;$
9   $for(i = 0; i < nr - 1; i++)\{$
10     $T' = ByteSub\_FM(T', r[l]);$
11     $T' = ShiftRow(T');$
12     $T' = MixColumn(T');$
13     $T' = T' \oplus K_i;$
14  $\}$
15  $T' = ByteSub\_FM(T', r[l]);$
16  $T' = ShiftRow(T');$
17  $T' = T' \oplus K_{nr};$
18  $T = ApplyMask2(T', l);$
19  $output\ T;$
```

Algorithm 10 $ApplyMask2$

```
$ApplyMask2(T', l)$
1   $(t'_{15}, t'_{14}, \ldots, t'_0) = T';$
```

```
2    $for(j = 0; j < 16; j++)\ t'_j = t'_j \oplus m[l];$
3    $T' = (t'_{15}, t'_{14}, \ldots, t'_0);$
4    $output\ T';$
```

Because the attacker cannot know which mask in the memory is used, he cannot decide the point for SODPA.

4.3 Consideration for Implementation

8-bit Microcontrollers like ST72101 has XOR instruction. The source operand can be accumulator and memory address. The destination operand can be accumulator. In implementing *ApplyMask*2, care should be taken so that mask value is not moved from the place where it is stored by line 3 or 4 in **Algorithm 9**. If it is moved to other memory, the moment can be a target for SODPA. If the mask is not moved and used in the place using the address of the mask, an attacker will not be able to find a point to mount SODPA.

5 Conclusions

It seems that fixed-value masking method is a good countermeasure for securing AES against DPA. The method is especially adequate for low cost smart cards that have little RAM and low processing power. But the method is vulnerable to SODPA and it can be more simplified maintaining security. In this paper, we propose a simplified SODPA resistant simple fixed-masking method. The proposed algorithm uses one byte for a mask and 256 bytes for modified S-box. It requires less ROM and processing. If we use 2 pairs of masks and modified S-boxes, required ROM or EEPROM will be 514 bytes. Although low cost smart cards have less RAM, they have relatively large ROM and EEPROM. Therefore the proposed method will be practical. By making it difficult to determine the point for SODPA, we achieved SODPA resistance. The additional RAM usage and processing for SODPA resistance is relatively small.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis", Advances in Cryptology - Crypto 1999, LNCS 1666, pp. 388-397, Springer-Verlag, 1999.

[2] T. Messerges, "Securing the AES Finalists Against Power Analysis Attacks", FSE 2000, LNCS 1978, pp. 150-164, Springer-Verlag, 2001.

[3] C. Clavier, J. Coron, and N. Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures", CHES 2000, LNCS 1965, pp. 252-263, Springer-Verlag, 2000.

[4] K. Itoh, M. Takenaka, and N. Torii, "DPA Countermeasure Based on the Masking Method", ICICS 2001, LNCS 2288, pp. 440-456, Springer-Verlag, 2002.

[5] M. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks", CHES 2001, LNCS 2162, pp. 309-318, Springer-Verlag, 2001.

[6] J. Golic and C. Tymen, "Multiplicative Masking and Power Analysis of AES", CHES 2002, LNCS 2523, pp. 198-212, Springer-Verlag, 2003.

[7] E. Trichina, D. Seta, and L. Germani, "Simplified Adaptive Multiplicative Masking for AES", CHES 2002, LNCS 2523, pp. 187-197, Springer-Verlag, 2003.

[8] T. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software", CHES 2000, LNCS 1965, pp. 238-251, Springer-Verlag, 2000.

[9] S. Yen, "Amplified Differential Power Cryptanalysis on Rijndael Implementations with Exponentially Fewer Power Traces", ACISP 2003, LNCS 2727, pp. 106-117, Springer-Verlag, 2003.

[10] Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", Crypto 2003, LNCS 2729, pp. 463-481, Springer-Verlag, 2003.